基于内容图像检索中的一种动态多维索引方法

徐 焕[1] 林坤辉[1] 周昌乐[2]
[1](厦门大学软件学院,福建厦门 361005)
[2](厦门大学信息科学与技术学院,福建厦门 361005)
E-mail:jploxh@126.com

摘 要 多维索引技术是基于内容检索的图像数据库的关键技术。SR-tree 和 X-tree 是目前比较成熟有效的多维索引技术。为了提高多维索引的性能,我们在分析 SR-tree 和 X-tree 的结构和性能的基础上,针对 SR-tree 分裂算法的不足,引入 X-tree 中超级节点的思想,通过改进插入和分裂算法,设计了一种新的多维索引结构 ESR-tree,即 Extended SR-tree。实验表明,随着数据量和维数的增多,ESR-tree 的性能明显优于 SR-tree 和 X-tree。

关键词 基于内容检索 多维索引 超球体 超矩形

文章编号 1002-8331-(2006)23-0161-04 **文献标识码** A **中图分类号** TP391

A Dynamic Multi-dimension Index Method in Content-based Image Retrieval

Xu Huan[1] Lin Kunhui[1] Zhou Changle[2]
[1](Software Department of Xiamen University, Xiamen, Fujian 361005)
[2](Information Science and Technology Department of Xiamen University, Xiamen, Fujian 361005)

Abstract: Multidimensional indexing technology is the key technology of content-based retrieval in image database.SR-tree and X-tree are riper and more effective multidimensional indexing technology at present.In order to improve the performance of the indexing, we assay the structure and capability of SR-tree and X-tree.In view of the deficiencies of split algorithm in SR-tree, the paper designs a new multidimensional indexing structure ESR-Tree, that is Extended SR-tree, which introduces the idea of super node of X-tree and improves the insert and split algorithm.The results of the experimentation show that the capability of ESR-Tree is better than SR-tree and X-tree with the increasing data number and dimensions.

Keywords: content-based retrieval, multidimensional indexing, super-sphere, super-rectangle

1 ESR-tree 的设计思想

基于内容检索的图像数据库一般数据量庞大,图像特征维度很高,为了提高检索效率,需要对图像特征建立高效的多维索引。采用不同的特征空间度量定义、包络类型和数据切分方法,可以得到不同类型的索引结构[1~3]。SR-tree[4,5]和 X-tree[6]是目前比较成熟有效的多维索引技术,SR-tree 在 SS-tree[7]的基础上引入了超矩形,采用超矩形和超球体结合的方法构造索引,减少了区域重叠,提高了区域之间的分离性,相对于 R*-tree[8],提高了最邻近查询的性能。但是 SR-tree 和 R*-tree 一样,也没有避免区域重叠。另一方面,除了在插入和删除操作后对包络的更新方法应同时更新超矩形和超球体外,SR-tree 的插入、删除以及分裂算法完全取自 SS-tree,超矩形信息在选择子树和决定分裂策略时不起任何作用;相反,对分裂策略起决定作用的超球体,却由于球体本身的几何特性,不大容易产生无重叠的分裂。针对 SR-tree 这一特点,我们利用 X-tree 中超节点减少重叠的思想[6],设计了一种新的索引结构 ESR-tree,即 Extended SR-tree,ESR-tree 是利用 X-tree 的设计思想来扩展 SR-tree,并改进了插入和分裂算法,在 ESR-tree 中通过引入超级节点,使得新的索引结构充分利用了 SR-tree 和 X-tree 两者的优点,弥补了 SR-tree 的不足,极大地减少了区域重叠,新的分裂算法能尽可能地产生无重叠分裂,有效提高了多维索引的性能。

2 ESR-Tree 的基本概念和主要算法

2.1 基本概念

定义 1 n 维空间中的矩形 R 可以由它的主对角线的两个顶点来定义:$R=(L, U)$。这里 $L=[l_1, l_2, \cdots, l_n]$, $U=[U_1, U_2, \cdots, U_n]$并且对于任何 i 都有 l_i $[U_i$。

定义 2 n 维空间中的球体 S 可以用它的球心和半径来定义:$S=(C,r)$。这里 $C=[C_1, C_2, \cdots, C_n]$是球心坐标,r 是球的半径。

定义 3 一个节点的超矩形包络 MBR 定义为包围该节点的所有子树中的点的最小边界矩形。

定义 4 一个节点的超球体包络定义为包围该节点的所有子树中的点的超球体,超球体包络是按下面的计算公式计算得

基金项目:国家 985 一般项目(编号:9850010)
作者简介:徐焕(1975-),女,硕士研究生,主要研究方向:网络多媒体,数据库应用。林坤辉(1961-),男,厦门大学计算机科学系副教授,主要研究方向:网络多媒体,数据库应用。周昌乐(1959-),男,教授,博士生导师,长期从事人工智能及其应用技术领域的研究工作。

到的，它并不是最小边界球体。

中心坐标 $X=(X_1, X_2, \cdots, X_n)$ 的计算方法：

$$\frac{\sum_{k=1}^{m} C_k.X_i * C_k.W}{\sum_{K=1}^{M} C_k.W}$$

$i=1, 2, \cdots, n$ 代表了数据空间的维数；

$C_k.X_i$ 代表第 k 个孩子在第 i 维上的中心坐标；

$C_k.W$ 代表第 k 个孩子的权重，也就是对应子树中的点的数量，子树只是一个叶子节点时，将叶子节点中的点看作是一个退化了的球体。这里 $k=1, 2, \cdots, m$ 是子树的序号。

半径 r 的计算方法：

$r=\min(ds, dr)$

$ds=\max(d(S_k, C))$

$dr=\max(d(R_k, C))$

C 是上面计算出来的球体的中心点。

定义 5 一个节点的包络定义为该节点的超球体包络和超矩形包络的相交区域：

$E=MBR \cap MBS$

2.2 节点结构

ESR-tree 的结构同 X-tree 的结构相似，它的总体结构由三类节点组成，即叶子节点、非叶子节点以及非叶子超节点[6]。

叶子节点的结构为：

$L: (E_1, E_2, \cdots, E_n)$

$E_i: (point, dataitem)\ (i=1, 2, \cdots, n)$

其中 point 是多维空间中的点（特征向量），dataitem 是同该点相关的数据，例如可以是该高维空间的点所代表的图像的文件名或者图像对象的对象标识 OID 。

非叶子节点的结构为：

$N: (C_1, C_2, \cdots, C_n)$

$C_i: (S, R, w, child\_pointer, split\_history, nodesize)\ (i=1, 2, \cdots, n)$

其中，child_pointer 是指向子树的指针；S 是 child_pointer 所指向的子树中所有点的包络球体，R 是子树中所有点的包络矩形；w 是子树中被索引点的数量；split_ history 是用于表示该入口项分裂历史的位向量，该位向量的第 i 位为 1，表明该入口项曾经以第 i 维为分裂维进行过分裂；nodesize 为该节点所占的外存页面数多少，nodesize>1 则说明该节点是超节点。

2.3 ESR-tree 的主要算法

插入算法和分裂算法是 ESR-tree 树中最重要的算法，它们最终决定了 ESR-tree 的结构，从而决定了 ESR-tree 的性能。ESR-tree 采用基于“中心点”的插入算法：当插入新数据时，在所有子树中选择“中心点”距离新数据点最近的那棵子树作为新数据点要插入的子树，递归进行直到找到适当的叶子节点。如果在插入过程中没有分裂，则逆序更新访问路径上的超矩形和超球体包络即可。若插入导致入口节点数上溢，则采用 SR-tree 的分裂算法和 X-tree 中超级节点的思想相结合来设计分裂算法。新的分裂算法能尽可能产生无重叠分裂，有效地减少重叠区域，提高空间利用率，进而提高了索引性能。以下主要给出插入时选择子树的算法和分裂算法：

算法 1 选择子树（chooseSubtree）

输入：新插入的点 point。

输出：被选择的入口项在节点中的序号（也就是子树的序号）。

步骤：

（1）计算 point 同节点中的每一个入口项的中心点的距离，假定距离最小的入口项为 entry[follow]；

（2）对该入口项的子树中多维点的个数值加 1；

（3）扩张超矩形，使新的矩形能包含 point；

（4）按加权平均的方式计算新的球体的中心点；扩张超球体，新球体的半径是中心点到所有子树的超矩形的最大距离和到所有子树的超球体的最大距离中的最小值；

（5）返回距离最小的入口项在节点中的序 follow。

该选择子树算法的优点是，在选择子树的过程中对包络球体和包络矩形进行更新，在实现上比较直观、容易。但这并不是包络更新算法的全部，当插入导致子节点分裂时，入口项本身的包络需要重新更新。

算法 2 分裂算法

算法 2-1 sphere-split

当插入导致入口节点数上溢时，首先采用强制重插策略[4]，将部分入口项删除并重新插入到树中。如果该节点上已经进行过强制重插，进行球体分裂，称之为 sphere-split。

输入：待分裂的节点。

输出：分裂所在的维，最佳分裂方案。

步骤：

（1）设定节点入口项数下限 minCount 的值：minCount=（maxCount*MINFANOUT）/100；（maxCount 为节点中入口项的最大个数，MINFANOUT 为节点扇出）；

（2）计算分裂所在的维：对每一维，计算所有入口项中超球体的中心点在该维坐标的方差，方差最大的维 dim 作为分裂所在的维[5]；

（3）选择最佳分裂方案：

①在分裂所在的维，按照中心点在该维坐标从小到大的顺序对所有的入口项进行排序；

②对所有的 totalEntries-minCount+1（totalEntries 是节点入口项总数）种可能的分裂方案得到的两个新节点，分别对每一维计算每个入口项的中心点在该维上坐标的方差，然后将 2*N 个方差值累加，这里 N 是数据空间的维数。方差之和最小的分裂方案就是最佳的分裂方案；

（4）返回 dim 和最佳分裂方案。

算法 2-2 rectangle-split

由于球体的几何特性，这种以提高空间利用率为目的的分裂策略极易导致包络区域之间较多的重叠。由于 ESR-tree 中的包络是超矩形包络和超球体包络的相交区域，所以，如果超矩形包络存在无重叠的分裂，则整个包络也一定存在无重叠的分裂。当 sphere-split 产生较多的重叠时，则可以利用超矩形包络的信息进行无重叠分裂，即 rectangle-split。

输入：待分裂的节点。

输出：分裂所在的维，最佳分裂方案。

步骤：

（1）设定 minCount 的值：minCount=1；

（2）将所有入口项的 split_history（入口项分裂历史的位向量）进行按位逻辑与操作，如果所得结果 split_vector 为 0，说明不可能有无重叠分裂，则直接退出，进行相应的生成超级节点[7]

的操作;

(3)选择最佳的无重叠分裂进行的维:split_vector中位为1的维都是无重叠分裂的维。对split_vector中位为1的每一维:

①将所有入口项中超矩形按矩形在该维的最小坐标值排序,并对所有的totalEntries-minCount+1种可能的分裂方案,计算分裂后的两个新节点的超矩形包络的边长之和;

②将所有入口项中超矩形按矩形在该维的最大坐标值排序,并对所有的totalEntries-minCount+1种可能的分裂方案,计算分裂后的两个新节点的超矩形包络的边长之和,二者之和最小的维dim就是分裂所在的维;

(4)选择最佳的分裂方案:

①对所有入口项中矩形分别按照在dim维的坐标最大值和最小值排序,得到两个超矩形序列;

②对所有的2*(totalEntries-2*minCount+1)种可能的分裂方案,分别计算出overlap,deadspace,它们分别代表两种排序方式下相应的两个超矩形包络之间的重叠和死区体积。重叠最小的分裂方案就是最佳分裂方案,重叠相同时则比较死区体积的大小;

(5)返回dim和最佳分裂方案。

上面的分裂算法中,对分裂维的选择要求所有的入口项很容易地分开,而对分裂方案的选择则使两个新节点中的入口项分别聚簇在一起。rectangle_split中minCount为1,也就是没有入口项个数的下限要求,这是产生无重叠分裂的要求,也是导致树结构不平衡的根源。当分裂导致树不平衡时,停止正常分裂,生成一个超节点;如果被分裂节点已经是超节点,则超级节点所占的块数增加1;

算法2-3 总的分裂算法

输入:待分裂的节点。

输出:如果分裂了,则返回指向新的节点的指针,同时设置分裂标志位。否则生成超级节点。

步骤:

(1)进行球体拓扑分裂sphere-split;

(2)判断新节点的超矩形包络之间的重叠,如果重叠小于预定的最大值,则跳到(5),否则跳到(3);

(3)进行超矩形无重叠分裂rectangle-split;

(4)如果生成的两个节点的扇出小于最小空间利用率的要求,则生成超节点,返回false,否则转(5);

(5)按照选定的分裂维和分裂方案进行分裂,生成两个新节点,更新入口项的分裂历史并返回true。

2.4 无重叠分裂存在性

由于在ESR-tree中对数据集的描述采用的是超矩形包络和超球体包络的相交区域,如果分裂后产生的两个节点的超矩形包络之间没有重叠,则该分裂必然是无重叠的分裂。所以只要证明分裂对超矩形而言是无重叠的即可。

定理 节点的分裂是无重叠分裂,当且仅当n维中存在一维,使得节点中的所有入口项都曾经在这一维上进行过分裂。

证明:

(1)必要性

假定对于每一维,至少存在一个入口项没有在该维上进行过分裂,则说明该入口项中的超矩形包络MBR在该维上包含了数据集的值域,为了不失一般性,假定该矩形为MBR1,相应的维为d。则在第d维上的分裂不可能是无重叠的。因为对于该节点中的任何其它矩形MBR,都有MBRHMBR1≡0。

(2)充分性

假定不可能存在无重叠的分裂。这就意味着不可能存在这样的维,使得可以在该维上将节点的入口项分为两个子集。这就同前提条件矛盾了。

3 测试结果及性能分析

3.1 测试条件

测试评价一个多维索引结构的性能需要一个合理的大规模测试数据集。对图像检索系统而言,一个好的测试数据集必须满足的条件是:(1)规模足够大,以测试检索系统中高维索引结构的可扩展性;(2)图像内容丰富,以测试图像特征的有效性和系统的总体性能。为了测试MPEG-7标准的有效性,MPEG-7专家组提供了一组测试集,包括音频、静止图像和视频三个部分。其中,静止图像部分包括7 000幅从各种渠道得到的各种格式的静止图像。但是由于该数据集的规模不够,不能很好地测试我们的索引结构的可扩展性。我们选用http://www.stb-ag.com/berchtol/papers/founew.norniert.gz上提供的特征数据集,该数据集曾用作X-tree的测试数据集,它的特点是特征向量是图像特征的付立叶系数表示,因此可以将特征向量的后若干维简单地去掉,仍然能够得到另一个有效的特征向量。该特征向量集的最大维数是32维,因此我们可以从中得到1到32维中任意维数的特征向量。

3.2 测试环境

本文中所做的所有测试使用的软硬件环境如下:

(1)硬件环境:PE1.2GHz CPU,256MB SDRAM内存,80GB硬盘,IBM兼容机。

(2)操作系统平台:Microsoft Windows XP。

(3)编程环境:Microsoft Visual C++6.0编译器。

3.3 测试方案

每个数据集上随机选出5个点,作为5个不同查询的查询输入点。每个查询执行10遍,查找出同查询输入点距离最近的8个点。由于所测试的索引结构都是基于外存的索引结构,因此内存和外存之间的I/O操作是影响查询性能的最主要的因素。而每次查询执行之后,查找路径上的节点都被读进了内存中,这样下次查询时,就减少了外存到内存的读操作,这样得到的查询响应时间就是所谓的热结果,但是这种热结果反应不出基于外存的索引结构的性能特点,因此我们只使用冷结果作为评价算法性能的依据:每次查询进行之后,都有相应的清理内存的动作。

3.4 测试结果分析

图1和图2显示了在维数一定的情况下(16维),索引结构的性能随着数据集的大小变化的曲线图。由两幅曲线图可以看出,I/O次数的变化趋势和CPU时间的变化趋势基本相同。这反应出受I/O次数约束的索引结构的一个特点:I/O时间占了CPU响应时间的大部分。如何从各个角度减少I/O次数成为设计索引结构的重要方面。数据集很小的时候(1万),各种索引结构的性能没有明显的差别。实际上,当数据集不大的时候,顺序查找也能提供很好的性能。上面的性能曲线图还说明,在I/O次数大致相同的情况下,ESR-tree需要更多的CPU时间。这是因为ESR-tree有更复杂的节点结构。当数据集很大的时候(5万),ESR-tree的性能优越性很明显,ESR-tree的查询

响应时间大概是 SR-tree 的 66.8%，是 X-tree 的 87.5%。

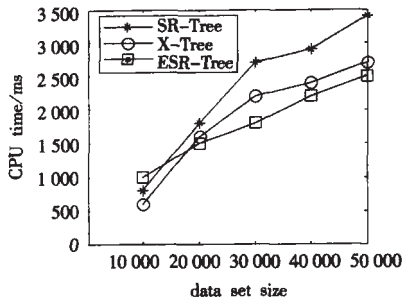

图 1 CPU 时间随数据集大小变化曲线

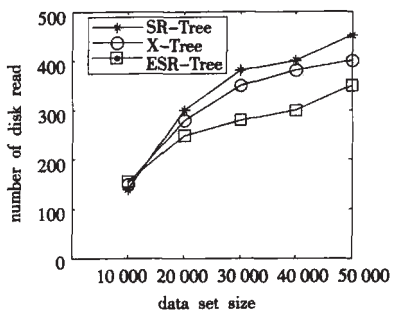

图 2 I/O 次数随数据集大小变化曲线

图 3 和图 4 显示了数据集大小一定的情况下(包含了 5 万个多维空间中的点)，索引结构的性能随着维数的变化而变化的曲线图。由图可以看出三种索引结构在数据空间维数较低(2 维、4 维)的时候表现出了几乎完全相同的性能。随着维数的增加，ESR-tree 和 X-tree 体现出了明显的性能优越性。在维数为 32 维的时候，ESR-tree 的查询响应时间分别是 SR-tree 的 67.78%和 X-tree 的 87.65%；而 X-tree 的查询响应时间是 SR-tree 的 77.34%。

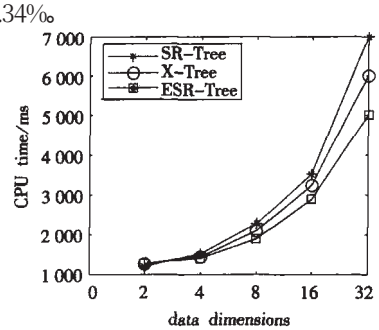

图 3 CPU 时间随维数变化曲线

4 结论

正如文献[9]中分析，X-tree 的性能要优于 SR-tree，但是我们的测试结果表明并没有数量级上的变化。X-tree 的查询响应时间大概是 SR-tree 的响应时间的 72.27%。由于 SR-tree 的各个子树的包络区域之间有较大的重叠，而 ESR-tree 的各个子树的包络区域之间的重叠则较少或者没有，重叠所造成的对多条路径的查找操作就会带来更多的 I/O 次数，所以 ESR-tree 需要更少的 I/O 操作。

上面的查询性能随着维数的变化曲线图还显示了这几种索引结构普遍存在的问题：查询性能随着维数的上升而下降。这反映出在索引结构的设计中应该注意的问题，如何降低数据空间的维数对查询性能的影响是今后的一个研究方向。

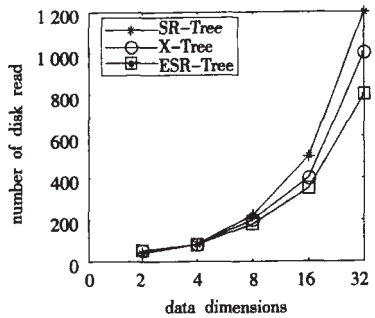

图 4 I/O 次数随数据维数变化曲线

ESR-tree 采用基于中心点的选择子树策略这种策略同 X-tree 以及 R*-tree 所采用的通过比较重叠区域的变化和超矩形体积的变化来选择适当的子树相比，需要更少的计算量。另一方面，ESR-tree 采用的是基于中心点的方差的节点分裂策略，这种策略同 X-tree 采用的根据超矩形在两种排序方式下的分裂后的边长、重叠大小和死区体积来决定分裂策略相比，也有更少的计算次数。

通过实验分析，ESR-tree 的性能比 X-tree 和 SR-tree 有所提高。数据集越大 ESR-tree 的优越性越明显。

(收稿日期：2005 年 11 月)

参考文献

1.C Bohm, S Berchtold, D A Keim. Searching in High-dimensional Spaces: Index structures for improving performance of multimedia databases[J]. ACM Computing Surveys, 2001; 33(3): 322~373

2.董道国，梁刘红，薛向阳. VAR-tree——一种新的高维数据索引结构[J]. 计算机研究与发展，2005; 42(1): 10~17

3.梅承力，周源华. 高维数据空间索引的研究[J]. 红外与激光工程，2002; 31(1): 77~81

4.Norio Katayama, Shin'ichi Satoh: The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries[DB/OL]. http://www.informatik.uni-trier.de/~ley/db/conf/sigmod/, 2002-02-02

5.N Katayama, S Satoh. The SR-tree: an index structure for high-dimensional nearest neighbor queries[C]. In: Proc of the 1997 ACM SIGMOD Intl Conf on Management of Data, 1997: 369~380

6.S Berchtold, D A Kieim, H-PKriegel. The X-tree: an index structure for high-dimensional data[C]. In: Proc of the 22nd VLDB Conf, 1996: 28~39

7.D A White, R Jain. Similarity indexing with the SS-tree[C]. In: Proc of the 12th Intl Conf on Data Engineering, 1996: 516~523

8.N Beckmann et al. R*-tree: An efficient and robust access method for points and rectangles[C]. In: Proc of the 1990 ACM SIGMOD Intl Conf on Management of Data, 1990: 322~331

9.K-I Lin, H V Jagadish, C Faloutsos. The TV-tree: An index structure for high-dimensional data[J]. VLDB Journal, 1994; 3(4): 517~542